

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Программирование алгоритмов с бинарными деревьями
Вариант 14-в

Студент гр. 8304

Чешуин Д. И.

Преподаватель

Фирсов М. А.

Санкт-Петербург

2019

Цель работы.

Познакомиться с часто используемой на практике нелинейной структурой данных – бинарным деревом. Изучить способы её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Постановка задачи.

Бинарное дерево называется бинарным деревом поиска, если для каждого его узла справедливо: все элементы правого поддерева больше этого узла, а все элементы левого поддерева – меньше этого узла.

Бинарное дерево называется пирамидой, если для каждого его узла справедливо: значения всех потомков этого узла не больше, чем значение узла.

Для заданного бинарного дерева с числовым типом элементов определить, является ли оно бинарным деревом поиска и является ли оно пирамидой.

Описание алгоритма.

Проверка осуществляется путём бинарного дерева на соответствие дереву поиска или пирамиде сравнения значения корня со значениями его левого и правого узлов(в случае дерева поиска левый узел должен быть меньше или равен, а правый больше). Если сравнение пройдено успешно, то таким же образом проверяется левое и правое поддерево. Проверки осуществляются до тех пор, пока все узлы не будут проверены. В случае, если дерево не соответствует дереву поиска, оно подобным же образом проверяется на соответствие пирамиде.

Спецификация программы.

Программа предназначена для проверки бинарного дерева на соответствие дереву поиска или пирамиде.

Программа написана на языке C++ с использованием CLI. Входными данными являются строки, соответствующие скобочному представлению бинарного дерева и введённые из консоли, либо из файла, переданного в качестве аргумента командной строки.

Тестирование.

Таблица 1 – Результаты тестирования программы

Input	Output
10	Search tree
10 (9) (11)	Search tree
10 (1) (2)	Pyramid
-1 (12) (25)	Simple tree
10 (9 (8) (7)) (6)	Pyramid
5 (1 (-1) ^) (8 ^ (9))	Search tree
5 ^ (10 ^ (-1))	Simple tree
10 (10 (10) (10)) (10 (10) (10))	Pyramid
-1 (1 (12) (3)) (165 ^ (18))	Simple tree
aaa	Invalid input
-1 (aa) (12)	Invalid input
10 (c (10) (b)) (10 (a) (10))	Invalid input

Анализ алгоритма.

Алгоритм работает за линейное время от размера дерева .

Описание функций и СД.

Класс **VBinaryTree** реализует структуру бинарного дерева на основе массива, а также методы для работы с ним.

Функция для проверки бинарного дерева на соответствие дереву поиска

```
bool isSearchTree(VBinaryTree<int>* bt);
```

Принимает на вход указатель на бинарное дерево и возвращает true – если дерево является деревом поиска и false в других случаях.

Функция для проверки бинарного дерева на соответствие пирамиде.

```
bool isPyramid(VBinaryTree<int>* bt);
```

Принимает на вход указатель на бинарное дерево и возвращает true – если дерево является пирамидой и false в других случаях

Выводы.

В ходе работы были приобретены навыки работы со бинарным деревом, изучены методы работы с ним. Был изучен алгоритм проверки дерева на соответствие дереву поиска и пирамиде.

Приложение А. Исходный код программы.

vbinarytree.h

```
#ifndef VBINARYTREE_H
#define VBINARYTREE_H
#include <stack>

template<typename T>
class VBinaryTree
{
private:
    struct Node
    {
        int parent_ = -1;
        int left_ = -1;
        int right_ = -1;
        bool isEmpty = true;
        T value_;
    };

private:
    Node* memory_ = nullptr;
    int maxSize_ = 10;
    int size_ = 1;

    std::stack<unsigned> freePoses_;
    int curPos_ = 0;
public:
    VBinaryTree();
    ~VBinaryTree();
    VBinaryTree(const VBinaryTree<T>& BT);
    VBinaryTree<T>& operator=(VBinaryTree<T>& BT);

    void setValue(T value);
    T getValue() const;
    size_t size() const;

    int toLeft();
    int toRight();
    int toParent();
    void toPos(int pos);

    int left() const;
    int right() const;
    int current() const;
    int parent() const;
    bool isEmpty() const;

    int addLeft();
    int addRight();
};

template<typename T>
VBinaryTree<T>::VBinaryTree()
{
    curPos_ = 0;
    memory_ = new Node[maxSize_];

    for(int i = 9; i > 0; i--)
    {
        freePoses_.push(i);
    }
}

template<typename T>
VBinaryTree<T>::~~VBinaryTree()
{
    delete[] memory_;
}

template<typename T>
VBinaryTree<T>::VBinaryTree(const VBinaryTree<T>& BT)
{
    curPos_ = 0;
    maxSize_ = BT.maxSize_;
    memory_ = new Node[maxSize_];
```

```

        for(int i = 0; i < maxSize_; i++)
        {
            memory_[i] = BT.memory_[i];
        }

        freePoses_ = BT.freePoses_;
    }

template<typename T>
VBinaryTree<T>& VBinaryTree<T>::operator=(VBinaryTree<T>& BT)
{
    curPos_ = BT.curPos_;
    maxSize_ = BT.maxSize_;

    if(memory_ != nullptr)
    {
        delete memory_;
    }
    memory_ = new Node[maxSize_];

    for(int i = 0; i < maxSize_; i++)
    {
        memory_[i] = BT.memory_[i];
    }

    freePoses_ = BT.freePoses_;

    return this;
}

template<typename T>
void VBinaryTree<T>::setValue(T value)
{
    memory_[curPos_].value_ = value;
    memory_[curPos_].isEmpty = false;
}

template<typename T>
T VBinaryTree<T>::getValue() const
{
    return memory_[curPos_].value_;
}

template<typename T>
int VBinaryTree<T>::toLeft()
{
    curPos_ = memory_[curPos_].left_;

    return curPos_;
}

template<typename T>
int VBinaryTree<T>::toRight()
{
    curPos_ = memory_[curPos_].right_;

    return curPos_;
}

template<typename T>
int VBinaryTree<T>::toParent()
{
    curPos_ = memory_[curPos_].parent_;

    return curPos_;
}

template<typename T>
void VBinaryTree<T>::toPos(int pos)
{
    curPos_ = pos;
}

template<typename T>
size_t VBinaryTree<T>::size() const
{
    return size_;
}

```

```

}

template<typename T>
int VBinaryTree<T>::left() const
{
    return memory_[curPos_].left_;
}

template<typename T>
int VBinaryTree<T>::right() const
{
    return memory_[curPos_].right_;
}

template<typename T>
int VBinaryTree<T>::parent() const
{
    return memory_[curPos_].parent_;
}

template<typename T>
int VBinaryTree<T>::current() const
{
    return curPos_;
}

template<typename T>
bool VBinaryTree<T>::isEmpty() const
{
    return memory_[curPos_].isEmpty;
}

template<typename T>
int VBinaryTree<T>::addLeft()
{
    size_ += 1;

    if (freePoses_.empty())
    {
        Node* buf = memory_;
        memory_ = new Node[maxSize_ * 2];

        for (int i = 0; i < maxSize_; i++)
        {
            memory_[i] = buf[i];
        }

        delete buf;

        maxSize_ = maxSize_ * 2;
        for (int i = maxSize_ * 2 - 1; i > maxSize_; i--)
        {
            freePoses_.push(i);
        }

        maxSize_ = maxSize_ * 2;
    }

    int buf = curPos_;
    int parent = buf;
    while (memory_[curPos_].left_ != -1)
    {
        parent = toLeft();
    }

    int newNode = freePoses_.top();
    freePoses_.pop();
    memory_[curPos_].left_ = newNode;
    memory_[newNode].parent_ = parent;

    curPos_ = buf;

    return newNode;
}

template<typename T>
int VBinaryTree<T>::addRight()
{
    size_ += 1;

```

```

    if (freePoses_.empty())
    {
        Node* buf = memory_;
        memory_ = new Node[maxSize_ * 2];

        for (int i = 0; i < maxSize_; i++)
        {
            memory_[i] = buf[i];
        }

        delete buf;

        maxSize_ = maxSize_ * 2;
        for (int i = maxSize_ * 2 - 1; i > maxSize_; i--)
        {
            freePoses_.push(i);
        }

        maxSize_ = maxSize_ * 2;
    }

    int buf = curPos_;
    int parent = buf;
    while (memory_[curPos_].right_ != -1)
    {
        parent = toRight();
    }

    int newNode = freePoses_.top();
    freePoses_.pop();
    memory_[curPos_].right_ = newNode;
    memory_[newNode].parent_ = parent;

    curPos_ = buf;

    return newNode;
}

```

```

#endif // VBINARYTEE_H

```

iomanager.h

```

#ifndef CLIHANDLER_H
#define CLIHANDLER_H

#include <iostream>
#include <fstream>
#include <memory>
#include <sstream>

class IoManager
{
private:
    int argc_ = 0;
    char** argv_ = nullptr;
    int curArgNum_ = 1;

    std::istream* curInStream_ = nullptr;
    std::ostream* curOutStream_ = nullptr;

    void openNextStream();
public:
    typedef std::shared_ptr<IoManager> IoManagerP;

    IoManager(int argc, char** argv);
    ~IoManager();
    std::istream* nextStream();
    void writeLine(std::string line);
};

#endif // CLIHANDLER_H

```

main.cpp

```

#include <iostream>
#include <string>
#include <queue>
#include "ioManager.h"
#include "vbinarytree.h"

```



```

VBinaryTree<int>* parse(std::istream& input);
bool isSearchTree(VBinaryTree<int>* bt);
bool isPyramid(VBinaryTree<int>* bt);

int main(int argc, char** argv)
{
    IoManager ioManager(argc, argv);
    std::string result;

    std::istream* stream = ioManager.nextStream();
    while(stream != nullptr)
    {
        std::getline(*stream, result);
        stream->seekg(0);

        VBinaryTree<int>* bt = parse(*stream);
        if(isSearchTree(bt))
        {
            result += " | search tree";
        }
        else if(isPyramid(bt))
        {
            result += " | pyramid";
        }
        else
        {
            result += " | simple binary tree";
        }

        ioManager.writeLine(result);

        delete stream;
        stream = ioManager.nextStream();
    }

    return 0;
}

VBinaryTree<int>* parse(std::istream& input)
{
    VBinaryTree<int>* bt = new VBinaryTree<int>();

    while(input.peek() != EOF)
    {
        std::string buf;
        input >> buf;

        if(buf == "(")
        {
            if(bt->left() == -1)
            {
                bt->addLeft();
                bt->toLeft();
            }
            else
            {
                bt->addRight();
                bt->toRight();
            }
        }
        else if (buf == ")")
        {
            bt->toParent();
        }
        else if(buf == "^")
        {
            bt->toParent();

            if(bt->left() == -1)
            {
                bt->addLeft();
                bt->toLeft();
            }
            else
            {
                bt->addRight();
                bt->toRight();
            }
        }
    }
}

```

```

        else
        {
            bt->setValue(stoi(buf));
        }
    }

    return bt;
}

bool isSearchTree(VBinaryTree<int>* bt)
{
    std::cout << "Checking on search tree." << std::endl;

    bool isSearch = true;
    int base = bt->current();

    std::queue<int> nodes;
    nodes.push(base);

    while(!nodes.empty() && isSearch == true)
    {
        bt->toPos(nodes.front());
        nodes.pop();

        int value = bt->getValue();

        std::cout << "Cur node value - " << value << std::endl;

        if(bt->left() != -1){
            bt->toLeft();
            if(!bt->isEmpty())
            {
                nodes.push(bt->current());

                if(bt->getValue() > value)
                {
                    std::cout << "Left node is bigger! - FAIL" << std::endl;
                    isSearch = false;
                    break;
                }
                std::cout << "Left node lesser or equal! - OK" << std::endl;
            }
            else
            {
                std::cout << "Left node is empty! - OK" << std::endl;
            }
            bt->toParent();
        }
        else
        {
            std::cout << "Left node is not exist! - OK" << std::endl;
        }

        if(bt->right() != -1)
        {
            bt->toRight();
            if(!bt->isEmpty())
            {
                nodes.push(bt->current());

                if(bt->getValue() <= value)
                {
                    std::cout << "Right node is lesser or equal! - FAIL" << std::endl;
                    isSearch = false;
                    break;
                }
                std::cout << "Right node bigger! - OK" << std::endl;
            }
            else
            {
                std::cout << "Right node is empty! - OK" << std::endl;
            }
        }
        else
        {
            std::cout << "Right node is not exist! - OK" << std::endl;
        }

        std::cout << "-----step-----" << std::endl;
    }
}

```

```

    }

    bt->toPos(base);

    return isSearch;
}

bool isPyramid(VBinaryTree<int>* bt)
{
    std::cout << "Checking on pyramid." << std::endl;

    bool isPyramid = true;
    int base = bt->current();

    std::queue<int> nodes;
    nodes.push(base);

    while(!nodes.empty() && isPyramid == true)
    {
        bt->toPos(nodes.front());
        nodes.pop();

        int value = bt->getValue();

        std::cout << "Cur node value - " << value << std::endl;

        if(bt->left() != -1){
            bt->toLeft();
            if(!bt->isEmpty())
            {
                nodes.push(bt->current());

                if(bt->getValue() > value)
                {
                    std::cout << "Left node is bigger! - FAIL" << std::endl;
                    isPyramid = false;
                    break;
                }
                std::cout << "Left node lesser or equal! - OK" << std::endl;
            }
            else
            {
                std::cout << "Left node is empty! - OK" << std::endl;
            }
            bt->toParent();
        }
        else
        {
            std::cout << "Left node is not exist! - OK" << std::endl;
        }

        if(bt->right() != -1)
        {
            bt->toRight();
            if(!bt->isEmpty())
            {
                nodes.push(bt->current());

                if(bt->getValue() > value)
                {
                    std::cout << "Right node is bigger! - FAIL" << std::endl;
                    isPyramid = false;
                    break;
                }
                std::cout << "Right node lesser or equal! - OK" << std::endl;
            }
            else
            {
                std::cout << "Right node is empty! - OK" << std::endl;
            }
        }
        else
        {
            std::cout << "Right node is not exist! - OK" << std::endl;
        }

        std::cout << "-----step-----" << std::endl;
    }
}

```

```

        bt->toPos(base);

        return isPyramid;
    }
}

iomanager.cpp
#include "ioManager.h"

IoManager::IoManager(int argc, char** argv)
{
    argc_ = argc;
    argv_ = argv;

    if(argc_ < 2)
    {
        curInStream_ = &std::cin;
        curOutStream_ = &std::cout;
    }
}

void IoManager::openNextStream()
{
    if(curInStream_ == nullptr){
        curInStream_ = new std::ifstream();
        curOutStream_ = new std::ofstream();
    }

    if(curArgNum_ >= argc_)
    {
        if(curInStream_ != &std::cin)
        {
            delete curInStream_;
            delete curOutStream_;
        }

        curInStream_ = nullptr;
        curOutStream_ = nullptr;

        return;
    }

    std::ifstream* inFileStream = static_cast<std::ifstream*>(curInStream_);
    if(inFileStream->is_open())
    {
        inFileStream->close();
    }
    std::ofstream* outFileStream = static_cast<std::ofstream*>(curOutStream_);
    if(outFileStream->is_open())
    {
        outFileStream->close();
    }

    while(curArgNum_ < argc_ && !inFileStream->is_open())
    {
        std::cout << "Try to open file - ";
        std::cout << argv_[curArgNum_] << std::endl;

        inFileStream->open(argv_[curArgNum_]);

        if(inFileStream->is_open())
        {
            std::string outFile(argv_[curArgNum_]);
            outFile += " - results";
            outFileStream->open(outFile);

            std::cout << "File opened." << std::endl << std::endl;
        }
        else
        {
            std::cout << "Can't open - file not founded" << std::endl << std::endl;
        }

        curArgNum_ += 1;
    }
}

std::istream* IoManager::nextStream()
{
    if(curInStream_ == nullptr)

```

```

{
    openNextStream();

    if(curInStream_ == nullptr)
    {
        return nullptr;
    }
}

while(curInStream_->peek() == EOF)
{
    openNextStream();

    if(curInStream_ == nullptr)
    {
        return nullptr;
    }
}

std::string buffer;

std::getline(*curInStream_, buffer);
if(buffer == "")
{
    return nullptr;
}

std::stringstream* sstream = new std::stringstream();
*sstream << buffer;

return sstream;
}

void IoManager::writeLine(std::string line)
{
    *curOutputStream_ << line << std::endl;
}

IoManager::~IoManager()
{
    if(curInStream_ != nullptr && curInStream_ != &std::cin)
    {
        delete curInStream_;
        delete curOutputStream_;
    }
}

```